Tia Kiesiläinen

# GAME DEVELOPMENT WITH THE FROZENBYTE ENGINE FOR THE GAME CONSOLES OF THE EIGHTH GENERATION

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Luonnontieteiden ala
Tietojenkäsittelyn koulutusohjelma
29.11.2014

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

**OPINNÄYTETYÖ**
**TIIVISTELMÄ**

| | |
|---|---|
| Koulutusala<br>Luonnontieteiden ala | Koulutusohjelma<br>Tietojenkäsittelyn koulutus |
| Tekijä(t)<br>Tia Kiesiläinen | |
| Työn nimi<br>Pelinkehitys Frozenbyte Oy:n pelimoottorilla kahdeksannen sukupolven pelikonsoleille | |
| Vaihtoehtoiset ammattiopinnot | Toimeksiantaja<br>Frozenbyte Oy |
| Aika<br>29.11.2014 | Sivumäärä ja liitteet<br>37 |


Tämän opinnäytetyön tilaaja on Frozenbyte Oy, joka on julkaissut aikaisempia pelejään monelle alustalle. Työn aihe valikoitui tekijän työharjoittelun aikana hänen työskennellessään Trine Enchanted Edition pelin parissa, ja valintaa tuki pelin lähestyvä julkaisu monelle alustalle sekä oppimismahdollisuus tekijälle. Tavoitteena oli antaa tekijälle tietoa ja käytännön kokemusta monen alusta tukemisesta pelimoottorissa ja konsoleille kehittämisestä. Työn aikana kehitettiin Frozenbyte Oy:n pelimoottoria ja lopulta julkaistiin peli.

Kahdeksannen sukupolven konsoleista julkaisualustoiksi valittiin PlayStation 4 ja WiiU konsolit. Valitut konsolit ovat suorityskyvyltään hyvin erilaisia, ja niiden kehitystyökalut eroavat myös toisistaan. Frozenbyte Oy:n pelimoottori on ohjelmoitu C++ -kielellä, joka tarjoaa erilaisia tapoja alustariippumattomuuden toteuttamiseen. Opinnäytetyön tekemisen aikana pelimoottorin olemassaoleva tuki PlayStation 4 ja WiiU -alustoille päivitettiin toimivaksi. Käytettyjä kolmannen osapuolen kirjastoja päivitettiin, ja pelimoottoria ja peliä muutettiin toimimaan paremmin valituilla alustoilla. Tekijä ja toinen ohjelmoija tekivät muutokset kokeneempien ohjelmoijien tukemina. Muutokset noudattivat konsolivalmistajien asettamia ohjeita ja rajoituksia, ja peliä myös testattiin jatkuvasti näiden ohjeiden mukaan.

Suurin työmäärä aiheutui ulkoisten kirjastojen, pelimoottorin, pelin ja alustojen yhteistoiminnan varmistamisesta. Pelimoottorista löytyi myös monia puutteita ja virheitä, jotka korjattiin. Työn loppuvaiheessa oli selvää, että monen alustan tukeminen vaatii pelimoottorin jatkuvaa ylläpitoa ja kehittämistä.


| | |
|---|---|
| Kieli | Englanti |
| Asiasanat | Pelimoottori, C++, ohjelmointi, pelikonsoli |
| Säilytyspaikka | ☒ Verkkokirjasto Theseus<br>☐ Kajaanin ammattikorkeakoulun kirjasto |

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

**THESIS ABSTRACT**

| School<br>School of Business | Degree Programme<br>Business Information Technology |
|---|---|
| Author(s)<br>Tia Kiesiläinen | |
| Title<br>Game Development with The Frozenbyte Engine for the Game Consoles of the Eighth Generation | |
| Optional Professional Studies | Commissioned by<br>Frozenbyte Ltd. |
| Date<br>29.11.2014 | Total Number of Pages and Appendices<br>37 |


This thesis was commissioned by Frozenbyte Ltd, who have a history of publishing games on multiple platforms. The subject was decided when the author worked on Trine Enchanted Edition during their internship. The decision was supported by the upcoming launch of the game for multiple platforms and the opportunities it allowed for learning.
The goal of the thesis is to further the author's knowledge and skills regarding development for multiple platforms. The practical goal is to further develop the engine and to publish the game.

Out of the eighth generation of consoles, PlayStation 4 and WiiU were selected as target platforms. Xbox One was left for later due to time constraints. The chosen consoles differ greatly on performance and available development tools. The Frozenbyte's engine is programmed in C++, which allows multiple ways to implement platform independence. During the development process the existing support for PlayStation 4 and WiiU was updated. Third party libraries used by the engine were also updated to the latest versions and the engine and game were tuned to work especially on the selected platforms.
The author and one other programmer did most of the work and were supported by senior programmers when needed. Changes to the engine followed the guidelines set by console manufacturers and the game was also tested to comply with the set requirements,

Most of the work went into making sure that the engine, game, external libraries and platforms worked together without errors. Numerous errors and shortcomings found in the engine were corrected. When the game was almost ready it had become apparent that platform independence requires constant upkeeping and development of the game engine.


| Language of Thesis | English |
|---|---|
| Keywords | Game engine, C++, programming, game console |
| Deposited at | ☒ Electronic library Theseus<br>☐ Library of Kajaani University of Applied Sciences |

# FOREWORD

To Erno.

TABLE OF CONTENTS

LIST OF SYMBOLS

| | |
|---|---|
| API | Short for application programming interface. A set of rules to which a software may conform. Using an API enables multiple software components to work together. |
| Build | A version of a software. During the development of a program, multiple builds are created and tested. |
| Development Kit | A custom game console that allows the execution of development builds. Development kits are regulated and distributed by the company which created the game console. |
| Development platform | A set combination of hardware and operating system, such as Windows computer, Nintendo WiiU console or an Android phone. |
| Game console | A development platform that is created and controlled by a private company. |
| Hardware | The physical components of a computer system. |
| IDE | Short for integrated development environment. A program which combines other programs needed to develop software from source code to an executable program. |
| Library | A collection of functions that are usable by other programs written in the same programming language. |
| Operating system | A software that controls the hardware of a system and allows multiple other programs to run concurrently and access the hardware in a controlled way. |
| Programming language | A programming language defines a syntax which needs to be followed to write valid programs on that language. |
| Source code | Code written by a programmer in a programming language. |
| Software | A single program, or a combination of programs. |

# 1 INTRODUCTION

Game engines reside at the heart of modern game development. Comprised of the most critical components for a game, they are used to supplying the foundation upon which the rest of the game is built. These engines evolve along the development of the game industry and platforms. As all software, they feature a score of different designs and patterns to fulfill their purpose. (Gregory 2009, 11.)

The subject of this thesis is to explain how one such engine has been put together and how it tackles one of the fundamental tasks of a modern game engine, that of offering support for publishing on multiple platforms. With an increasing number of available platforms and thus a growing audience that is available for a game, combined with the ease of digital distribution systems, publishing a game to multiple platforms has become increasingly profitable. With the advent of new technology and game engines, it has also become more feasible than ever before. Even though the platforms have become more similar to each other, the quirks and features that set them apart are still vital.

A successful game engine thus needs to be able to keep up with the times and simultaneously afford a solid foundation for the development of games.

The following chapters feature the fundamentals of a generic game engine and the different ways to deal with the challenge of supporting multiple platforms. A closer look is taken into the Frozenbyte engine, which is central for the goal of this thesis. This thesis will also present the specific way in which it deals with multiple platforms. An introduction is then given about the eighth generation of game consoles and their attributes as a target platforms for game development.

After all the theory, the goal of the thesis project is established and detailed. A plan of action is formed, based on the theory in earlier chapters. Execution and results of the plan are presented next. The last chapter gives an analysis about further development in the areas discussed.

The idea for this thesis came about, as the author did their internship at Frozenbyte Ltd, a long standing game development company located in Helsinki, Finland. The company utilizes their own custom built game engine and are known for publishing to multiple platforms. The eighth generation of game consoles had just arrived to the market and the company was eventually going to publish their most beloved Trine series to these platforms. Author's interest was piqued, owning to the fact that however complete their

education had been in game development that far, the subject of game consoles and the challenges of multiple platforms had been not featured in the curriculum. Luckily the interest coincided with the company's to familiarize new developers to the secrets of the Frozenbyte engine. The development and publishing of Trine Enchanted Edition offered a learning opportunity for the author and other junior programmers.

As we will see further on, this kind of opportunity is rather rare for students and even for small companies, as the powers behind the most known game consoles have strict requirements for the aspiring developers and thus none of the hardware or documentation is freely available for use or study.

The twofold goal of the thesis is thus to improve the understanding and practical skills of the author and to make this understanding most beneficial to the company now and in the future.

# 2 GAME ENGINE

As game engines are meant to provide the fundamental building blocks for a game, they all contain the same basic components. The most essential is of course the rendering engine. This part of the game engine is responsible for actually drawing the game on the screen. Other basic parts include, but are not limited to, the collision and physics engine, animation system and audio system. Collision and physics take care of detecting interaction and collision between game objects. Animation system is used to animate the game objects, which can be either two dimensional images or three dimensional models. The audio system plays all the sounds and music in the game. The game objects themselves are herded by a game world system, that keeps track of the creation, updating and deletion of those objects. (Gregory 2009, 3.)

All these systems contribute to create a simulation of the game world that is running in real time and responds to the player's actions. This simulated world is populated by game objects and agents. The latter are game objects directed by artificial intelligence, which in turn is handled by the artificial intelligence system. Depending on the type of the game, this simulated world can be close to the natural one or as imaginative as the developers can make it. So it is important to note that the game engines do not attempt to create a facsimile of the real world, but a world following the game's rules. Different kind of programs exist for scientific simulation purposes. (Gregory 2009, 9.)

The simulation of this world happens inside a ”game loop” which activates each of the systems in turn or in parallel and they carry on to handle their respective tasks and update their part of the simulation. When one iteration of the loop is completed, the result is rendered to the player. The game loop is run multiple times in a second, which gives an impression of a continuous system. The sounds and music are played over multiple loops and the player input is inspected at least once during each iteration. (Gregory 2009, 10.)

## 2.1 Game engine basics

A game engine is the foundation for a new game. The development of a game starts with a clean engine, but during the development the two are usually mixed together at least in some degree. A game engine can cater to a wide range of game genres or be genre

specific (Gregory 2009, 12). Other distinction is whether the game engine is commercially available to multiple developers or if it is an in-house engine.

Game engine handles the general and fundamental tasks needed by the game. It offers an interface to the different aspects of the platform and rudimentary things such as playing audio and drawing the graphics.

The purpose and benefit of using a game engine is to speed up development and to reduce errors. As the game engine provides the necessary grounding, the developers do not need to invent the wheel each time a new game project is started and all improvements in the game engine benefit all the games made with it. All these things allow more time to be used on the actual game mechanics and assets.

The first notion of a ”game engine” appeared in the wake of the first-person shooter game Doom by id Software. Doom was made with a well defined separation between the core systems of rendering, collision detection and audio from the game assets, levels and game play rules (Gregory 2009, 11). This enabled extensive and easy modification of the game and later game series called Quake from the same company took advantage of a similar design. The company also sold the Quake engine as a separate product that was licensed to other game developers. The other developers who bought a license would receive the engine and build their own game around it. It is typical for game engines to be first developed inside a company for their own use and later licensed to other companies or released as open source software (Munro, Boldyreff & Capiluppi 2009, 1).

Despite the many advantages of such design, it is not yet a universal custom to have a clearly separate engine. Games today represent a broad spectrum in the level of separation between the core engine components and game specific parts. The basic trade-off is that with an engine it is easier and faster to develop content, but without a set engine, the game can be better optimized to fit certain requirements. (Gregory 2009, 11.)

What is more, game engines are often made to cater for requirements of a specific genre. For example an engine made for a two person boxing game is suited towards representing accurate motions in a confined space with high level of detail and speed. On the other end, a game engine for a strategy game is concerned with AI, rendering the terrain and keeping track of all the units on the field. However different, all engines require the same facilities such as rendering, handling input and so on. The implementation and focus of these systems varies with the intended purpose of the engine, but their purpose remains the same. (Gregory 2009, 13.)

One more unifying aspect between modern game engines is that the core systems are invariably written in either C or C++, as these closely related languages are geared towards the same things that the core systems are. They are at home with accessing low level system components and allow the best size against performance ratio. The other contributing factor to their use is the fact that almost all external libraries are written in these languages and their use is thus easiest from a C/C++ system. On the higher level a game engine may employ a scripting language, but this varies from engine to engine. As the foothold of C languages is so strong in the field of the game engines, this thesis will give examples of key points using C++. If the reader is not familiar with this language, the basics are presented in the following section.

## 2.2 Introduction to C++

C++ is a general purpose, systems programming language. It hails from the 1970's and today it is widely used and in continuous development. The most important feature in C++ regarding game engines and multiple platforms is its ability to work very closely with the hardware of the platform. Unlike some other languages, C++ gives direct access to memory locations and the data they contain. The compilers of C++ produce very efficient code and are able to automatically optimize code to run on a given platform. Also, the innumerable libraries already written in C++ help greatly in creating a new engine. Other defining feature is the ability to include source code based on given variables. This allows the C++ compiler to only include the code that is necessary. We shall see the usage and benefits of this feature in chapter 3. (Stroustrup 2013, 9.)

As a downside, C++ has long compilation times, which takes time away from the day to day development. The problem is made worse by the fact that a change in a piece of code also requires a recompilation of all code that is using the modified code and that some integrated development environments are unable to detect simple syntax errors without compilation. In the worst case a small typing error in a critical location causes a long compilation progress, which then results in an error and wasted time.

The long compilation times are caused by the build progress of the C++, which has multiple phases before the final executable program can be run. This progress is explained in illustration 1 below. (Pakkanen 2012).

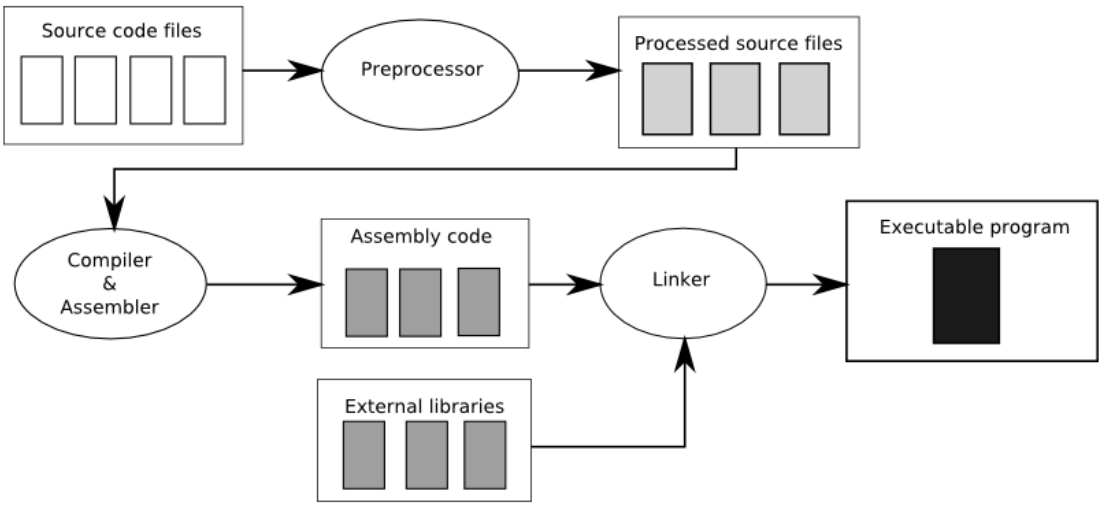


*Figure 1: C++ build progress (Stroustrup 2013, 38).*

The source code files written by programmers are first processed by a preprocessor, which replaces the macros with their values, removes comments and leaves out parts of code based on conditions and defines. Each of these source code files is then compiled by the compiler, which together with the assembler turns the written code into object files containing machine language, that are only valid for a single platform. The compiler is able to detect which files have not been modified since the last compilation. If the file has not been updated, the previously generated object file can be used. In the last phase these object files are linked together with each other by the linker program. It finds and sets addresses for function calls in the written code and in the external libraries that are included in the program. The final result is the executable program. Any of these phases may fail, which results in an error message. A C++ programmer needs to know these messages as some of them can be rather cryptic and not clearly indicate what actually went wrong. Each of the phases can be controlled by multiple settings. For example the compiler can be instructed to optimize code in such way that the final executable takes less space on disk or to have the final executable to be large, but more efficient. (Gregory 2009, 72.) (Stroustrup 2013, 420.)

## 2.3 The Frozenbyte Game Engine

The Frozenbyte Game Engine is an in-house engine, and its different versions have been used to developed all the games that the company has published. Most recent version was made for the Trine 2 and is currently in use. The engine is programmed in C++ and uses

Lua as a scripting language. It makes use of third party extensions such as PhysX physics simulation library by NVidia, Speex voice compression library by Xiph.org and Wwise audio system and audio editor by Audiokinect.

The engine is very modular and made of successive layers, that offer services to the upper and more abstract portions. The engine features numerous wrapper interfaces that hide the implementation details of third party libraries and allow a different set of libraries to be used on different platforms. There is an automated tool which checks for dependencies between layers of the engine and issues a warning if a layer is dependent on a layer situated higher up in the hierarchy.

Essential to developing games with the Frozenbyte engine is the editor, which is used to create the levels of the Trine games and to adjust the components defined in the engine to suit each particular situation. The editor also allows for the creation of types, which are custom collections of components working together. The editor itself is written in C# and works only on the Windows platform. Other development tools include a performance statistics visualization program, data archiving program and a build automation system.

The engine has been used to create multiple games. The first one was Shadowgrounds, which was a top down, three dimensional shooter game. It was followed by Shadowgrounds Survivor and then the first Trine game. Trine was followed by a sequel which also had an add-on called Goblin Menace. Between releases the engine has been constantly developed. The game that is presented in this thesis is the first Trine game being redone in the current version of the engine and called Trine Enchanted Edition.

# 3 GAME CONSOLES OF THE EIGHTH GENERATION

The game consoles of the eighth generation include the WiiU from Nintendo, PlayStation 4 from Sony Entertainment and Xbox One from Microsoft. Additionally, several micro-consoles can be counted as a part of the eighth generation, but they are notably different and thus not discussed here. (Engadget 2013.)

All the consoles are closed proprietary systems and development for them requires a contract with the related company. Developers are classified as independent developers, third party developers or as a first party developers. Independent developers apply for a new contract for each published game. Third party developers have a longer standing contract with the company or the company has approached them with a contract of making a game for the platform. First party developers are directly managed by the console manufacturer and develop games exclusively for their console. (Edge 2014.)

To help in development, the company provides documentation, support services and development kits, which are modified versions of the console. The development kit and related software enable the developer to test their game on the console while it is still in development. The developer is also bound by technical requirements that state how the game is to behave on the platform and places restrictions on the memory and processor usage. (Edge 2014.)

## 3.1 PlayStation 4

PlayStation 4 is the fourth console in the successful line of PlayStation consoles by Sony. The first PlayStation was launched in 1994, and the second version is one of the most sold game consoles (Guardian 2013). The PlayStation's direct competitor is the Xbox line of consoles and in the eighth generation the fierce competition continues, which started with the release of PlayStation 3 and Xbox 360 in the previous generation. PlayStation 4 was launched in November 2013 (PlayStationLifeStyle.net 2013).

PlayStation 4 is built around an AMD Jaguar microprocessor and unlike its predecessor, is very close to the architecture of a gaming PC (Taylor 2013). This design makes it easier for game developers to develop for the platform. The core chip features an accelerated processing unit (APU) which combines together a central processing unit

(CPU), graphical processing unit (GPU), memory controller and video encoder/decoder. The chip also houses a secondary custom chip that is used for various undemanding background tasks. This double chip design allows the background tasks to be run without affecting the performance when playing (Conditt 2013).

The central processing unit has two modules with 4 cores each, which allow for 8 processes to run simultaneously. PlayStation 4 has 8 gigabytes of memory which is shared between the graphics processing unit and the central processing unit. This eliminates the delays in data transfer between the CPU and GPU (Mujtaba 2012). Of this memory, 5 gigabytes are available to be used by the currently running game and rest is reserved for the operating system (Eurogamer 2013).

Games can be installed from the Sony marketplace through an internet connection or from Blu-ray discs. They can also be updated using the internet connection.

Independent development is supported by Sony by removing unnecessary steps and allowing studios to self publish and set the prize for their games (The Verge 2014).

## 3.2 WiiU

WiiU is the successor to the Wii console, which was the first one to introduce motion based controls. WiiU follows this idea and also adds a gaming style called asynchronous multiplayer, which is made possible by the WiiU controller having a touch capable display. This display is independent and thus the player using the controller can see information that other players can not. WiiU was released in 2012 (Metro 2012).

Like Wii, the WiiU tries not to compete with the other consoles in terms of performance but takes a different approach focusing on the ease of use, familiarity and game selection that is geared towards the whole family. It offers backwards compatibility for Wii games (Nintendo 2014).

WiiU also relies on a chip produced by AMD, but the similarities to competitors end there. It is a multi-chip module featuring a PowerPC central processing unit with three cores. The graphics chip has one graphical processing unit for WiiU games and one for Wii games. Similar to the PlayStation 4, there is also a slower background chip and a dedicated hardware audio module for processing signals. (fail0verflow 2014.)

The console has 2 gigabytes of memory of which 1 gigabyte is reserved for the operating system. Rest of the memory is shared between the CPU and GPU as in PlayStation 4 (fail0verflow 2014).

The special feature of the WiiU is the selection of different controllers. The main controller is the WiiU game pad with a touch screen display and traditional controls. Other controllers feature the remote and nunchuck pair and a pro controller and a classic controller (Fingas 2012).

## 3.3 Xbox One

Xbox One is the direct competitor of the PlayStation 4. The differences are subtle as both employ a desktop computer architecture and mostly same parts. The differences are in the marketing, intended use and peripherals. Xbox One is targeted first and foremost to North American market and was launched there and in Australia, New Zealand and several European markets first in November 2013. Rest of the world got the Xbox One almost a year later in September 2014. This is a different strategy to the other consoles which were launched almost simultaneously to all markets. (Skillings 2014.)

What comes to hardware, the Xbox One is almost identical with the PlayStation 4. It features almost the same accelerated processing unit of the Jaguar architecture. The difference is the number of cores in the graphical processing unit. Both consoles have the same amount of memory, albeit in Xbox One the memory operation speed is slower. As in PlayStation 4, 3 gigabytes of memory are reserved for the operating system and the rest are meant for the game (Shimpi 2013). They also share the Blu-ray drive.

The special features of Xbox One are the Kinect motion detecting controller and voice control interface. Kinect can be used for controlling the games and to do additional tasks such as scanning QR codes. Voice control can be used to activate functions of the console. (O'Brien 2013.)

The console also features a SmartClass technology, which allows a suitable mobile phone to be used to control the console and act as a secondary screen (Plafke 2013).

## 3.4 Comparison of hardware

The differences in the hardware between the platforms of the eighth generation can be seen in table 1. It is clear that where the PlayStation 4 and Xbox One have only minor differences, the WiiU is very different from both of them. When developing a game for all three platforms, a decision must be made to either conform to the lowest performance or to adjust the quality and content of the game according to the platform. All the properties listed in the table 1 have an effect on the game and game development.

| | PlayStation 4 | WiiU | Xbox One |
|---|---|---|---|
| Available memory | 5 of 8 gigabytes | 1 of 2 gigabytes | 5 of 8 gigabytes |
| Memory type | DDR5 | DDR3 | DDR3 |
| CPU architecture | x86-64 | PowerPC | x86-64 |
| CPU cores | 8 | 3 | 8 |
| CPU speed | 1.6 GHz | 1.24 GHz | 1.75 GHz |
| GPU architecture | Jaguar | Radeon R600/R700 | Jaguar |
| GPU cores | 1152 | 400 | 768 |
| GPU speed | 800 MHz | 550 MHz | 854 MHz |
| Disc drive type | Blu-ray dual, DVD | WiiU optical disc | Blu-ray, DVD, CD |
| Storage size | 500 gigabytes | 8 gigabytes | 500 gigabytes |

Table 1: Comparison of console hardware. (Shimpi 2013.) (fail0verflow 2014.) (Mugdal 2012.)

### Available memory

This is the amount of random access memory (RAM), that is usable by the game while it is running. It limits how many textures, models, sounds and other assets can be loaded simultaneously and be readily available when needed. If there is not enough memory, the quality of the assets has to be lowered or the game world needs to be partitioned. Partitioning requires loading screens, during which the memory is populated with the

assets needed. It is also possible to try to predict what assets are going to be needed next and continuously stream them from the hard drive into the memory. (Gordon 2013.)

CPU architecture

Architecture of the processor is a minor factor, but it might force the developer to use a different compiler for each architecture. The biggest difference here is that the WiiU is a 32-bit system while both PlayStation 4 and Xbox One are 64-bit systems. The main difference is the amount of memory that the processor can access. A 32-bit processor can only access 4 gigabytes of RAM memory. (Brain 2014.)

CPU cores

Having more than one core allows for concurrent programming, where calculations are made in parallel. As all the consoles have multiple cores, this technique can be used on all of them. Naturally, having a higher number of cores allows for more calculations to happen simultaneously. An important part of the game engine design is to utilize concurrency as much as possible. (Binstock 2008.)

CPU speed

Speed of the central processing unit is an indication of how many operations it can accomplish in a second. If the unit has multiple cores, all the cores function at the same speed. The difference in speed is not great across the platforms and the number of cores has a far greater effect on the performance. (Gordon 2011.)

GPU architecture

All graphics units of the eighth generation are manufactured by AMD and belong to the Radeon series. Both PlayStation 4 and Xbox One employ the Jaguar architecture, with their own modifications. The architecture used in WiiU is called GX2, and it is based on the consumer market Radeon R600 and R700 designs (fail0verflow 2014).

GPU compute units.

As with CPU's, the graphical processing unit also has multiple cores, which are grouped together into compute units. As with multiple cores, this value indicates how many tasks can be done concurrently. It is important to note that the graphical processing units in PlayStation 4 and Xbox One are similar to each other and WiiU is again different, as it is

based on much older hardware. So the number of cores is not the only thing creating a difference between the graphical processing power of the platforms. (Smith 2011.)

GPU speed

The speed of the GPU is noticeably slower than that of the CPU, but this is more than countered by the number of cores.

Disc drive type

While all the eighth generation consoles have a digital marketplace for buying games, most games for the consoles are sold on physical discs. The type of disc controls how much data the game can have. The Blu-ray disc can contain 25 gigabytes in single layer configuration and 50 gigabytes in dual layer configuration. WiiU uses a specially made disc which can contain up to 25 gigabytes of data and is similar to single layer Blu-ray discs (Gilbert 2012).

Storage size

This is the amount of permanent storage space for games and other files. Some games install all or part of their data to the hard disk for faster loading times and digitally purchased titles have to be stored on the console itself. Here again WiiU is significantly different from the others.

# 4 PLATFORM INDEPENDENCE IN GAME ENGINES

Platforms differ from each other in multiple ways. They use different libraries for the common tasks such as audio and graphics. Platforms might have varying amount of available memory and other resources at the game's disposal or they might force restrictions and constraints to the game's behavior. Especially with game consoles, they might even employ a different processor architecture than the computer on which the game engine is running on.

The independence from platforms means that a piece of code produces the same result regardless of the platform. Ideally most of the code falls into this category. Other parts of the code need to be platform specific in turn, to enable the general code to function.

If a game engine is made to be platform independent, this goal must be taken into account across the engine. The independence from platforms is constructed from features of the programming language and engine design that allows those abilities to be used. This chapter discusses central issues of both types and how they work with the rest of the engine.

## Atomic Data Types

In a programming language, atomic data types are those built-in types that form the basis for the user defined types and structures. Each atomic type is intended for a different purpose and to store different kind of information. Examples of atomic data types in C++ are char, int and float. Char type is used for storing a single character, int is intended for integer numbers and float for floating point or decimal numbers. Each has a different size in bytes. The C++ standard does not define absolute sizes for these types, instead they are defined with bounds that are relative to the size of char. The char most often takes up one byte and an int can be of same size or larger. Exact size of char is platform specific. (Stroustrup 2013, 138.) (Gregory 2009, 103.)

This leads into problems when the engine excepts something to be of certain size or to have a certain maximum value. A solution is to define custom types inside the engine that take up the same number of space on every platform. (Gregory 2009, 104.)

## Collections and iterators

Collections and iterators are provided by the C++ standard library and include different data structures. They are a common feature also in other programming languages. They have the same problems that are encountered with atomic types. The type and memory usage of collections might vary between platforms, leading to performance differences (info.prelert.com 2014). While not as fundamental as the difference between atomic types, it is convenient for the engine have platform independent versions of needed collection structures and their iterators in a way that is demonstrated in the following chapter. The structures in the engine can be based on the standard structures or they can be built from the ground up.

Threading Library

Especially with the eighth generation, a game can take advantage of the multiple cores on the system. To do this efficiently is an art in itself. In an ideal situation all the cores are active. It is the job of the programmer and the used threading library to divide the workload evenly across multiple cores. The amount of cores and threads can be known on consoles, but gaming computers can have from 2 to 12 cores and thus it is best to have a scalable system that can employ any number of cores. Essential to threading is the concept of locking, which locks a piece of information so that only one process at a time can modify it. The implementation of this lock can vary between platforms and some methods might be more efficient than others. (Gregory 2009, 324.)

Graphics

The two main standards for drawing graphics in games are the OpenGL and DirectX libraries (Abi-Chahla 2008). The consoles however might differ from this and have their own rendering library. All the graphic libraries are concerned with the same tasks and the game engine can have an abstraction layer that passes the drawing calls to the system used on the detected platform.

Physics

Physics system calculates the positions of all the dynamic game objects and notices when they collide with each other or how they should be affected by forces. Prominent physics libraries include PhysX by Nvidia, Havok Physics by Havok and open source library Bullet (Gregory 2009, 602). Some developers write their own physics system. The physics engine can be tuned to have a certain resolution. Lower resolution means that the

update steps are longer and the collisions are not as detailed. This is faster to calculate, but can lead to errors in the simulation especially when objects move very fast or there are multiple objects colliding with each other (Gregory 2009, 620).

## 4.1 Platform independence layer

Essential for the platform independence is the platform independence layer, which hides all the details of the platform from the rest of the engine. It also wraps the standard library functions of the C++ language or provides a custom implementation and abstracts the calls to the platform's operating system. This is to ensure consistency across the use of different implementations and to ease the use of the engine from the programmer's point of view. The platform independence layer is presented in figure 2, using audio as an example.

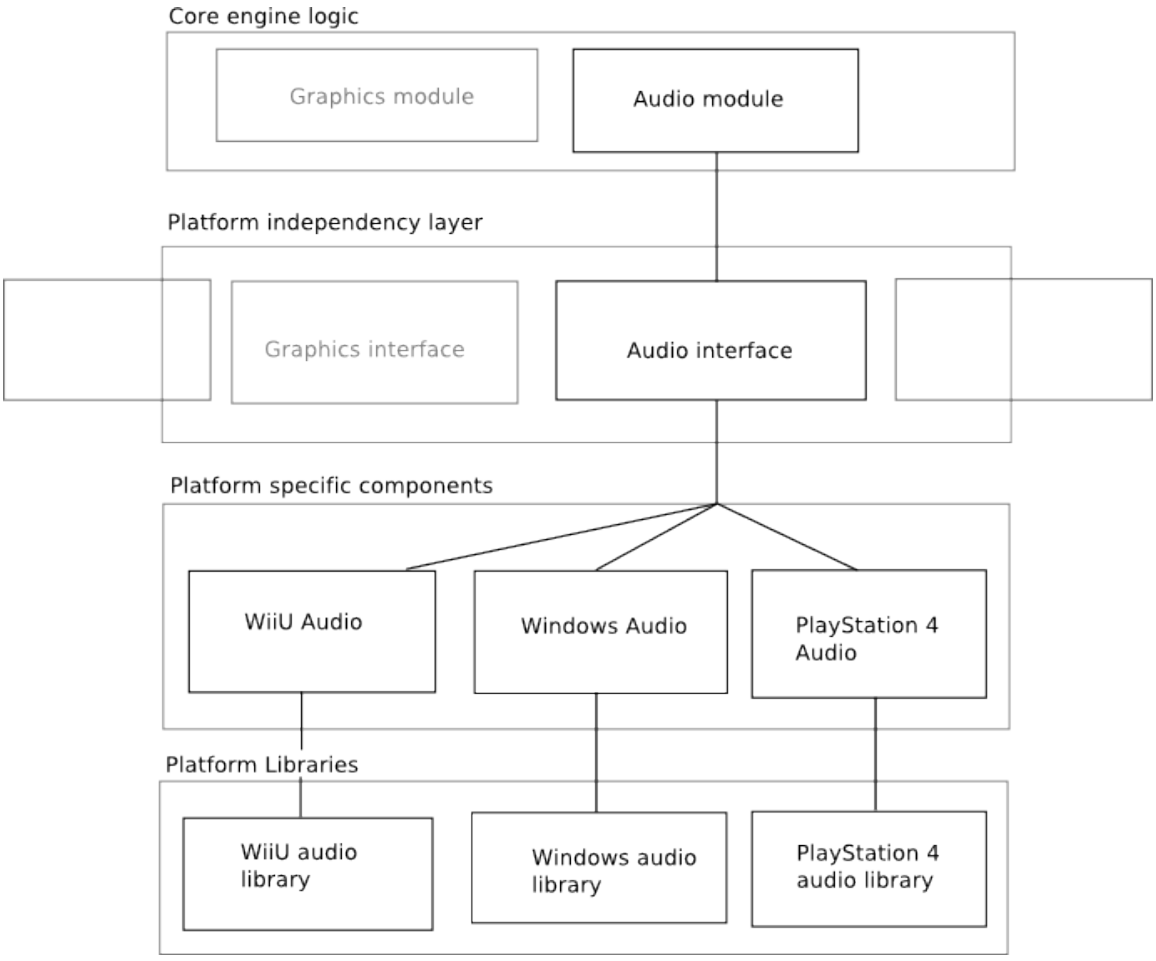


*Figure 2: The role of the platform independence layer (Gregory 2009, 34).*

The audio module of the engine uses the audio interface in the platform independence layer, which hides the details of the implementation from the rest of the engine. Depending on the target platform, this interface uses the correct component, which conforms to the interface of the audio system in the engine. This component then translates the instructions from the interface to the platform specific library and reports any errors and other messages back to the engine. Note that all the platforms might use the same library, as is the case with the Frozenbyte engine using Wwise, but still they would need to have separate components as the hardware and performance of the platforms is different.

## 4.2 Different approaches to platform independence

How the general and platform specific parts of the code are made to work together is an issue of game engine design. A simple way is to write general code and when need arises, to write specific code that is enabled depending on the platform used. While straightforward, this approach mixes up the two and spreads specific code all over the engine. This makes the engine difficult to maintain if a platform is added or an existing platform is changed.

More structured approach is to identify the platform dependent sections and create interfaces that link together the general code and the platform specific code. In this way the specific details are abstracted away and the code remains more clear. The downside is that identifying and building the needed interfaces takes time. The platform differences can also be handled on a higher level, by setting up the project so that it includes files and code based on the target platform.

A third way entails using a language that is interpreted on the platform itself and the virtual machine or the interpreter takes care of the platform specific issues. The drawback is the need to create such interpreter and the reduced performance of such approach. The language Java uses this approach to deal with multiple platforms.

## 4.3 Example of implementing platform independence in C++

In this example we use a syntax similar to C++. The most important thing to understand is the notion of a global definition. Such definition is shared and visible for each source file. Such defines are often simple boolean values, meaning they are either on or off. In this example, each platform has a corresponding definition, that is set to on when the code is generated for that platform. Logically, only one such definition can be on at any given time. Let us have two platforms, Windows and WiiU, whose defines shall be PLATFORM_WINDOWS and PLATFORM_WIIU.

Identifying the target platform with a preprocessor variable is similar to the technique used in C++ to identify the compiler to the preprocessor. For example the Microsoft compiler defines _MSC_VER. This can be used by the programmer to toggle compiler specific features. This is helpful as some platforms might have dedicated compilers. (Gregory 2009, 71.)

Let us use an example function for playing a sound, that is platform dependent, as different platforms have different libraries for playing the sound. Both platforms have their own function for playing a sound, namely playSoundWin() and playSoundWiiU().

The most straightforward case would be to choose between platforms each time a sound is played:

```
#ifdef PLATFORM_WINDOWS
        playSoundWindows();
#endif

#ifdef PLATFORM_WIIU
        playSoundWiiU();
#endif
```

The #ifdef is a keyword for the C++ preprocessor, that processes the code before it is compiled. The above usage would only leave one of the function calls in the code that is compiled. However simple this method is, its drawback becomes apparent when the program grows in size and occasions for playing a sound multiply. And if one would want to add a third platform, all those occasions would have to be found and modified to include the third option.

A more elegant way is to define a general function called playSound() that is platform independent and inside this function use the code written above. Then the programmer only needs to call the function playSound() whenever a sound is needed, without worrying about the platform. Similar approach could be taken with all sound related functions. This is an example of a platform independence layer, that hides the specific implementations behind a general functionality.

One could still go further and gather all sound related functions used by a platform to a single source file and name them with platform independent names. In the file for WiiU, the function playSoundWiiU() would be called playSound(). Each platform would have a similar file. If all these files would be included in the project, the compiler would be at loss to understand which functions to use, as the files contain identical names. The programmer would have to specify in the project settings that on the event of building for the WiiU, the file for the WiiU should be included. In this way all the implementation regarding one platform would be contained in a single file. This might not sound as a much of an improvement compared to the previous solution, but it adds one more layer of abstraction and enables an easy way to update the implementation of a single platform without worrying about the others.

## 4.4 Platform independence in the Frozenbyte Engine

The Frozenbyte engine uses both first and third of the three approaches discussed earlier. The platform specific code is sprinkled among the general code using preprocessor #if statements that change what pieces of code are included in a build. They are also used to define what libraries are linked with the application. Each platform has it's own specific define, and one of them is always set as FB_PLATFORM. A piece of code intended for WiiU is surrounded with:

```
#if FB_PLATFORM == FB_WIIU
… code ...
#endif
```

This technique is used especially in controls related code, as every platform has a different controlling scheme. It is also used with threading and memory, as platforms have different processors and memory management. In rendering code it is used to select the library used by the platform, usually either OpenGL or DirectX.

The GUI system of the engine uses both C++ and Lua. C++ classes and functions provide the low level functions for drawing windows and buttons. Individual menus and their behaviour is programmed in Lua. As Lua is not a compiled language, it cannot use same techniques. Instead, Lua has a component called platformManager, which has knowledge of the current platform. This allows Lua to create the menu windows differently for each platform by asking the platformManager which platform is in use.

The third approach of platform specific files is only used for some high level parts. For example the interface between the game engine and the operating system of a console resides in a platform specific file.

The editor is able to emulate different platforms to some degree. While it cannot undo the actions done by C++ preprocessor, it can change the platform in the Lua's platformManager. This enables developers to test functionality of GUI and menus without using the development kit or compiling the source code with a different platform define.

# 5 GOAL OF THE THESIS

The goal and purpose of this thesis is to gather information about the development process of Trine Enchanted Edition on the Frozenbyte engine. The game is made for Windows platform and the eighth generation of the game consoles and the special requirements and solutions of this process are the focus in the action phase of the thesis. Acquired information can then be used by future projects to avoid problems and pitfalls.

Secondary goal is to increase author's understanding and expertise of developing to game consoles. These skills will be valuable both to the author and Frozenbyte Ltd. in the future. Tertiary goal is to witness and work with the actual publishing procedure, but due to time constraints and delays, this goal might not be accomplished.

## 5.1 Specifications of the project

The author is working directly with the porting project and solving the issues that arise in the development. The senior programmers provide advice and help with tackling more difficult problems as necessary. The game is to work with acceptable frame rate, which means 30 frames per second for the WiiU and 60 frames per second for PlayStation 4. Game needs to pass the requirements of both Nintendo and Sony and also clear the internal testing process of Frozenbyte. Deadline for publishing the game is December 2014.

## 5.2 Initial setting for the project

Frozenbyte has previously ported games to WiiU and PlayStation 4 and so the requirements and the procedure is known, but the code that was used back then has been altered and a solid solution that covers all the platforms simultaneously is still missing. The platforms themselves have also changed since then, as are many of the third party libraries used by the engine. The code also contains numerous quick and dirty solutions to previously emerged problems and figuring and smoothing them out relies heavily on the expertise and memories of the senior developers.

## 5.3 Plan for reaching the goal

The plan for the porting project is to first release the Windows version of the game on the Steam platform to eliminate all possible game play related bugs and issues. When the game works in Windows as wanted it is safe to start working on other platforms. The WiiU and PS4 platforms will be handled first, in that order.

Depending on the relations with Microsoft the Xbox One version can also be done. Hopefully the lessons learned during porting to the other consoles will speed up the process.

## 5.4 Trine Enchanted Edition

Trine Enchanted edition is a game belonging to the platform genre. The platformer genre is right at home with the consoles, with two of the Japanese giants Nintendo and Sega both having their platformer characters, Mario and Sonic, as the company mascots (Lee 2011). Due to its accessibility and clear game play, the platformer genre has enjoyed lasting success through all console generations. Today, the three dimensional game world is the standard, and platformer games have been comfortable with it from early on. Some of these games fall more to the action game genre, but many of them still have the jumping mechanisms at the core of the game play. (Gamasutra 2006 a.)

A platformer game requires certain things from the engine. Moving platforms, elevators, dynamically moving objects and other pieces of interactive environment require the collision and physics engine. Puzzles and their elements require an easy way to link objects together and to launch new events when conditions are met. An editor and a scripting system are needed to link the objects. It can also be done with C++ programming, but it is tedious and time consuming to create and adjust. A camera must be clever enough to follow the player around and in case of a free movement in 3D environment, it needs to be either scripted or intelligent enough to always show the game from such angle that the platforms and their distance to the player is easy to understand. The camera can also be controllable by player (Gregory 2009, 363).

If the game contains enemies or other agents, the AI system needs to guide them in the world and be able to react convincingly to players presence and actions. The other

alternative for the agent behavior is to make them predictable and function more like moving obstacles, like is done in the two dimensional Super Mario games.

Trine Enchanted Edition is played like a two dimensional platformer, but the levels are made of three dimensional objects. It focuses on physics based puzzles and navigating the environments which are laden with traps, collectibles, platforms and enemies (Frozenbyte 2014). The biggest challenges for the engine are the intelligent enemies and the lush visuals that need to be optimized for the game to run smoothly. The other challenge not visible to the player is the editor and the underlying component based system, that are used to build the levels in the game.

Games in the Trine series are rather ideal for game consoles. They feature a co-operative mode, which lets up to three players to enjoy the adventure together. The main difficulty in the games are the puzzles presented by the environment, spiced by occasional battles against enemies. The difficulty can be set to suit players of different skill levels and the detailed visuals offer eye candy to onlookers. Owning to the two dimensional representation, the games can utilize best of rendering techniques with a consistent frame rate, as the viewing angle into the game world is limited. This helps to narrow the gap between consoles of different performance levels. The games also offer replay value with the aforementioned difficulty settings, multiple secrets and unrestricted solutions to the puzzles. The modern consoles also offer a digital distribution systems which allows Frozenbyte to keep selling the games on an agreeable price, compared to the boxed games that often may be asking up to 50 euros due to the extra costs which come with the distribution chain.

# 6 IMPLEMENTATION OF THE PROJECT PLAN

Seeing that most of the work was already done in the form of the engine support for WiiU and PlayStation 4 when Trine 2 was developed, it was hard to estimate the amount of necessary changes. The issue was more about how the changes made to the engine after the release of Trine 2 would work with the updated WiiU and PS4 software development kits and libraries and updated external libraries.

## 6.1 Changes made to the Frozenbyte engine and their effects

Multiple changes were made and they ranged from simple fixes and error corrections to rewriting and rewiring entire subsystems. Below are collected the most noteworthy areas where changes occurred.

### Audio changes

The Frozenbyte engine uses the Wwise audio middleware, which offers direct support for the WiiU system. During development the Wwise libraries were updated to the latest version 2014.5. It also uses the Speex library for voice transmission over internet. There was a new version of it available, called Opus, but the decision was made to not to upgrade to limit the amount of changes needed.

Updating the audio libraries fixed a serious bug that was occurring when the game transferred itself to background state to allow the WiiU menu to show. Upon returning to the foreground the audio would not continue. This was fixed by an update of the Wwise library. However, the new version of the Wwise library was not compatible with the Speex library and the players were not able to talk with each other over the internet. The Wwise was restored to the version used with Trine 2 and the serious bug was handled by a senior programmer.

### Threading

Threading was changed in a way that moved physics related tasks to run on all cores and centered critical operations on one core. This was to increase the speed of the physics calculations and to avoid different parts fighting over the same core.

Changes in threading and core affinity increased the frame rate considerably, but it also caused problems with the WiiU. The WiiU requires the game to only use a single core when it is in the background state. When the game is moved to background, all threads need to be relocated to this single core and this transition caused PhysX threads to often lose or corrupt the data they were working with. A senior programmer was able to mend this issue by introducing more control over threads and controlling when it was safe for them to share resources and data. This prevented data corruption efficiently.

Controller interface

The controller interface of the Frozenbyte engine hides the specifics of a controller from the rest of the engine. All controllers show up as buttons and axes, whose state can be monitored by the game. Each platform is handled by its dedicated human input device interface. The touchscreen of the Nintendo WiiU game pad does not convert to this system so easily and extra work was needed to make the buttons on the touch screen to work. The system was also generally improved by changing it to event based from a function based design. Each input event causes an event which is then inspected by the code.

Video

Previously the engine used Bink video. Due to high licensing costs, it was changed to Theora that is able to display Vorbis encoded videos. The memory handling of Theora is much less advanced than the streaming solution of Bink. This caused game to crash as the memory ran out as the whole video was loaded into memory. Solution was to decrease the quality of the videos. For the PS4 the ffmpeg library was used. With the video system change the cost of development decreased and the engine was less reliable to a proprietary third party library.

PhysX update

Version 3 of PhysX had been published after Trine 2 and the engine were updated to this new version. This caused a number of central functions to deprecate. The design of the engine is such that all the physics interface calls are handled through a wrapper but some parts remained that called PhysX functions directly. These calls were updated to the new version. Changing out the deprecated functions increased stability on the engine and increased the speed of the physics update.

### Resource management and loading

Previously the engine was only developed and used for Trine 2 and thus all the available resources such as textures, models, fonts and so on were expected to be in the game. With the introduction of Trine Enchanted Edition this changed. While the two games shared most of the resources and assets, many more of them were game specific. However, the engine did not support differentiating between different games and many of the resources were included in the game that were never used. This became apparent when the WiiU console failed to load certain maps into the system memory due to the memory being already populated by unneeded assets. It took some time to figure out the source of the problem, as the memory can also run out because of the game is reserving too much memory or leaking memory. Leaking of memory happens when a program reserves space from memory but does not mark it as freed when that space is no longer needed. The extra data in the memory remains and cannot be automatically detected and removed. The change here was to formulate a new system which would include the resources based on their usage in the included maps. This saved the enormous trouble of identifying and modifying the properties of the assets by hand. Still, some assets had to be identified and removed by hand. With the changes, the memory load went down considerably, allowing the game to work with every map and also in multiplayer.

### Linking

Linking is the phase in the C++ build process when the individually compiled source files are linked together by creating an address space for the program and placing the code and functions inside this space. When a function needs access to a function declared in another object file, the address of that function is located in the address space and given to the file needing it. This caused problems on the WiiU, because the linker and compiler used with WiiU are different from the ones used with PlayStation 4 or Windows. As the engine and game are composed of numerous layers, which have numerous dependencies between each other, the linker had to keep all the function addresses in the memory simultaneously, as any of them might be needed by any function. The development computers did not have enough memory for this and the linker could not function. Because of the differences in linkers, this only happened when linking the WiiU build. A solution was to divide the project into smaller independent parts, that would be linked as dynamic libraries. Using dynamic libraries, the linker is able to limit the amount of memory needed, as the libraries are only used by other files,

but the library is already linked to everything it needs. This proved to be very difficult and a senior programmer was assigned to this task. Other tweaks were also made to diminish the memory usage of the linker. Few low level containers were streamlined and template functions were removed when possible. Changes to the code made linking possible and also enabled the use of a new version of the toolset.

## 6.2 Quality assurance and publishing the game on the platforms

The game was tested rigorously by the quality assurance department of Frozenbyte. Aside from the normal game play testing, it was also checked against the requirements of console manufacturers. These requirements are listed in a confidential document that specifies numerous things about the game's allowed behavior, such as loading times, internet connection behavior, usage of sound, controller vibration and so on. Each item is provided with a test case and requirements for passing.

When the game passes the internal testing, it is sent to the manufacturer for further testing. According to the results the game needs to be updated until it passes all tests. Only after then can it be published. Publishing also requires setting up the distribution system and providing images and text that are used on the console's digital marketplace.

# 7 REFLECTION AND FURTHER DEVELOPMENT

Personally the project turned out to be very different than I imagined. The kind of problems encountered were not so much about low level differences between the platforms, but most often a question of how to use the third party libraries, how to write platform suitable code and how to optimize for each platforms strengths and weaknesses.. The latter holds true especially on WiiU as the system's performance differs greatly of that offered by PlayStation 4. During development the performance difference between the two systems was a mixed blessing. On the other hand, the memory and performance constraints on WiiU caused multiple problems and required extra work to make the game function smoothly. On the other hand, this process exploited multiple shortcomings and lazy solutions in the engine, which were then repaired and improved upon for the benefit of the engine and future games.

As the engine had been already used to publish games on both WiiU and PlayStation 4, most of the requirements were already fulfilled by the engine, but odd changes were still needed and some of them were tricky to catch and understand. This clearly showed how multiplatform support cannot be set in stone, as the platforms are constantly evolving. Added to this is the evolution of external libraries, which may put them out of sync with each other and the platforms.

During the project numerous suggestions for further development were voiced. Increased support for concurrency in all areas of the engine was one of the most recurring suggestions. It would shorten all loading times and take more advantage of modern multicore systems. Also a rewrite of the low level rendering system to better support a wider range of systems was seen as necessary and was scheduled to be started after Trine Enchanted Edition was ready. Working with the graphical user interface also required too much time and effort compared to the complexity of the needed changes and the whole system is going to be replaced before any user interface is added to future games. During the project I became the unofficial GUI specialist and would be the default programmer to handle any problems and changes related to the menu and other systems that were made in Lua. Those included the timing of videos and related sounds, playing the correct music for the levels and making sure that the menus worked on all platforms and in all languages equally well.

As the project neared completion, the old saying that work is half done when 90% of the work is done ringed true to me. There was always so little left, but those last bugs and errors turned out to be very complicated and new errors were found by quality assurance or caused by changes every day. As with every game project that I have been part of, in the last moments some features had to be cut and others were passed as good enough. It is with all creative endeavors that there is always a possibility to make things better or to refine some aspects, but there is simply not enough time. But there are always new games and new opportunities which can benefit from the errors of the past.

REFERENCES

Abi-Chahla F. 2008. http://www.tomshardware.com/reviews/opengl-directx,2019.html (Read 15.11.2014).

Binstock A. 2008. https://software.intel.com/en-us/articles/multi-core-processor-architecture-explained (Read 15.11.2014).

Brain M. 2014. http://computer.howstuffworks.com/microprocessor6.htm (Read 15.11.2014).

Conditt J. 2013. http://www.joystiq.com/2013/02/20/ps4-allows-playing-games-as-theyre-downloading/ (Read 15.11.2014).

Edge. 2014. http://www.edge-online.com/features/sony-microsoft-and-nintendos-indie-pitch-inside-ukies-developer-day/ (Read 15.11.2014).

Engadget. 2013. http://www.engadget.com/discuss/what-defines-a-new-generation-of-game-consoles-1lot/ (Read 15.11.2014).

Eurogamer. 2013. http://www.eurogamer.net/articles/digitalfoundry-ps3-system-software-memory (Read 2.11.2014).

fail0verflow. 2014. https://fail0verflow.com/blog/2014/console-hacking-2013-omake.html (Read 15.11.2014).

Fingas J. 2012. http://www.engadget.com/2012/06/05/nintendo-two-wii-u-gamepads-will-work-on-one-system/ (Read 15.11.2014).

Frozenbyte. 2014. http://www.frozenbyte.com/games/trine/ (Read 19.11.2014).

Gamasutra a. 2006. http://www.gamasutra.com/view/feature/1851/a_detailed_crossexamination_of_.php (Read 10.9.2014).

Gilbert B. 2012. http://www.engadget.com/2012/11/12/nintendo-wii-u-proprietary-disc/ (Read 15.11.2014).

Gordon W. 2013. http://lifehacker.com/when-ram-speed-matters-and-how-it-affects-your-games-1436679680 (Read 15.11.2014).

Gordon W. 2011. http://lifehacker.com/5796846/why-clock-speed-doesnt-matter-much-when-comparing-two-computer-processors (Read 15.11.2014).

Gregory J. 2009. Game Engine Architecture. Wellesley: A K Peters, Ltd.

Guardian 2013. http://www.theguardian.com/technology/2013/jan/04/playstation-2-manufacture-ends-years?INTCMP=SRCH (Read 25.9.2014).

info.prelert.com. 2014. http://info.prelert.com/blog/stl-container-memory-usage (Read 19.11.2014).

Lee D. 2011. http://www.bbc.co.uk/news/technology-13874266 (Read 15.11.2014).

Metro. 2012. http://metro.co.uk/2012/01/26/wii-u-confirmed-for-europe-this-year-297126/ (Read 15.11.2014).

Mugdal K. 2012. http://gamingbolt.com/wii-u-ram-is-43-slower-than-ps3360-ram (Read 15.11.2014).

Mujtaba H. 2012. http://wccftech.com/playstation-4-specifications-analysis-similar-cost-pc/ (Read 15.11.2014).

Munro J, Boldyreff C & Capiluppi A. 2009. Architectural Studies of Games Engines – the Quake Series. London: Games Innovations Conference, 2009. ICE-GIC 2009. International IEEE Consumer Electronics Society's.

Nintendo. 2014. https://www.nintendo.co.uk/Wii-U/Hardware-Features/Backwards-compatibility/Backwards-compatibility-736682.html (Read 19.11.2014).

O'Brien T. 2013. http://www.engadget.com/2013/05/21/microsofts-new-kinect-is-official/ (Read 15.11.2014).

Pakkanen J. 2012. http://voices.canonical.com/jussi.pakkanen/2012/10/01/building-cc-what-really-happens-and-why-does-it-take-so-long/ (Read 15.11.2014).

Plafke J. 2013. http://www.geek.com/games/e3-2013-how-the-xbox-one-will-use-smartglass-1558263/ (Read 15.11.2014).

PlayStationLifeStyle.net. 2013. http://www.playstationlifestyle.net/2013/08/20/playstation-4-release-date-confirmed-for/ (Read 15.11.2014).

Shimpi A. 2013. http://www.anandtech.com/show/6976/amds-jaguar-architecture-the-cpu-powering-xbox-one-playstation-4-kabini-temash/4 (Read 15.11.2014).

Skillings J. 2014. http://www.cnet.com/news/xbox-one-to-hit-26-new-countries-in-september/ (Read 15.11.2014).

Smith R. 2011. http://www.anandtech.com/show/4455/amds-graphics-core-next-preview-amd-architects-for-compute/5 (Read 15.11.2014).

Stroustrup B. 2013. The C++ Programming Language Fourth Edition. Ann Arbor: Addison-Wesley.

Taylor J. 2013. http://community.amd.com/community/amd-blogs/amd-gaming/blog/2013/02/21/amd-and-the-sony-ps4-allow-me-to-elaborate (Read 15.11.2014).

The Verge. 2014. http://www.theverge.com/2013/3/29/4162430/sony-playstation-indie-games-love-story 4.8.14 (Read 5.9.2014).